Computer Languages, Systems & Structures 46 (2016) 66–90

Contents lists available at ScienceDirect

Computer Languages, Systems & Structures

journal homepage: www.elsevier.com/locate/cl

Lightweight and static verification of UML executable models

Elena Planas [a,*], Jordi Cabot [a,b], Cristina Gómez [c]

[a] Universitat Oberta de Catalunya, Spain
[b] ICREA, Spain
[c] Universitat Politècnica de Catalunya, Spain

ARTICLE INFO

Article history:
Received 22 September 2015
Received in revised form
7 July 2016
Accepted 7 July 2016
Available online 19 July 2016

Keywords:
Model-Driven Development (MDD)
Model-Driven Architecture (MDA)
Executable models
Verification
Static analysis
Alf action language

ABSTRACT

Executable models play a key role in many software development methods by facilitating the (semi)automatic implementation/execution of the software system under development. This is possible because executable models promote a complete and fine-grained specification of the system behaviour. In this context, where models are the basis of the whole development process, the quality of the models has a high impact on the final quality of software systems derived from them. Therefore, the existence of methods to verify the correctness of executable models is crucial. Otherwise, the quality of the executable models (and in turn the quality of the final system generated from them) will be compromised. In this paper a lightweight and static verification method to assess the correctness of executable models is proposed. This method allows us to check whether the operations defined as part of the behavioural model are able to be executed without breaking the integrity of the structural model and returns a meaningful feedback that helps repairing the detected inconsistencies.

1. Introduction

Executable models are models with a behavioural specification detailed enough so that they can be systematically implemented or executed in the production environment. Executable models play a cornerstone role in the Model-Driven Development (MDD) paradigm, where models are the core artifacts of the development life-cycle and the basis to generate the final software implementation.

Executable models are not a new concept (e.g. [31,58]) but are now experiencing a comeback, becoming a relevant topic within the OMG (Object Management Group). Their use is being promoted given the value they can bring. As one of its creators declares "*executable models increase productivity by raising the level of abstraction*; *reduce costs by describing systems independently of their implementation*; *and improve the quality of the final system by facilitating early verification*" [37]. Following this trend, the OMG has published in the last few years several versions of the *Foundational Subset for Executable UML Models* (fUML) standard [46], an executable subset of the UML that can be used to define, in an operational style, the semantics of systems. The OMG has also published the first standard version of the *Action Language for Foundational UML* (Alf) standard [45], a textual surface notation which maps to the fUML [46], that provides the constructs to specify the fine-grained behaviour of systems in terms of actions. As can be seen, the increasing implementation of these standards in modeling tools [19,39,11] shows a raising interest for this topic.

* Correspondence to: Rambla del Poblenou 156, 08018 Barcelona, Spain. Fax: +34 93 326 88 22.
E-mail addresses: eplanash@uoc.edu (E. Planas), jordi.cabot@icrea.cat (J. Cabot), cristina@essi.upc.edu (C. Gómez).

http://dx.doi.org/10.1016/j.cl.2016.07.002
1477-8424/

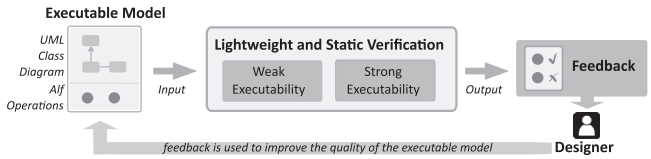

Fig. 1. Method overview.

Given the growing importance of executable models and the impact of their correctness on the final quality of software systems derived from them [41], the existence of methods to verify the correctness of such models is becoming crucial. Otherwise, the quality of the executable models (and in turn the quality of the final system generated from them) will be compromised.

In this paper we propose a method (see Fig. 1) focussed on the verification of the *executability* of operations specified by means of actions (i.e. action-based operations) with respect to a subset of the integrity constraints that can appear in a model. We consider that this is important criteria to preserve the correctness of such models. Besides checking the executability of the operations, the method we propose returns a meaningful feedback that helps repairing the detected inconsistencies.

Like other studies that focus on executability [12,13,55], our method classifies the operations in three categories:

1. *Strongly executable* (*SE*) *operations*, i.e. operations that are guaranteed to *always* generate a consistent state. We know for sure that all executions of the operation (regardless of the input values provided to the operation and the initial system state where the operation is applied on) reach a consistent state with respect to the structural model and their integrity constraints.
2. *Weakly executable* (*WE*) *operations*, i.e. operations that *sometimes* generate a consistent state, but are not guaranteed to do so. We can ensure that at least one of the many possible executions of the operation during the life span of the system will be successfully executed but probably not all of them (e.g. depending on the input parameters).
3. *Non-executable* (*¬E*) *operations*, i.e. operations that *never* generate a consistent system state. After their execution, they always reach a state that violates some integrity constraints of the structural model (e.g. some cardinality constraints). Note that this does not mean that the operations cannot be executed, but that their execution always generate an inconsistent system state.

¬ E operations, when executed as standalone operations, are useless since every time a user tries to execute them (regardless of the provided input values) an error arises because some integrity constraints become violated. Also notice that weak executability is a necessary (but not sufficient) condition for strong executability. Then SE operations are a subset of WE operations.

In contrast with most related works, our method follows a *lightweight* approach. The term *lightweight* was popularized over almost twenty years ago following the publication of a round-table article [56], which invited the use of formal methods in a practical way. We consider our method as being lightweight since it performs a static analysis of the model, i.e. it examines the model without executing its operations [40]. Static analysis has been mainly applied to analyze the source code of programs [54], but the same idea may be extended to analyze models at design time without requiring any kind of simulation. Besides, our method provides valuable information as feedback (in case of error, it points out the source of the error and assists the designer during their correction). As we will see later, our method relies on some assumptions regarding the input model and requires some trade-offs to enable our lightweight analysis.

The work reported here extends our previous works [51,52] on several directions: (1) we increase the expressiveness of the models that our method can deal with; (2) in order to align our work with the new UML standards, we now specify the operations by means of the Alf action language [45] provided by the OMG; (3) we re-design our previous methods [51,52] to provide a unique and integrated method for verifying weak and strong executability properties; (4) in our previous work [52] we only provide the skeleton of the method as a black box, while in this work we elaborate in detail each step of the method; and (5) we provide a prototype tool that implements our method.

To the best of our knowledge, only few works have been devoted to verify fUML/Alf specifications [4,8,17,33,35,38]. As suggested in [42,24] and more specifically in [49], the verification of such specifications should be studied.

Paper organization: The rest of the paper is structured as follows. Section 2 presents the motivation of our work. Section 3 introduces the basic concepts and the notation that will be used in the rest of the paper. Section 4 defines the concepts of *execution paths* and *executability*. Section 5 describes our method, Section 6 presents the prototype tool that implements it, Section 7 describes an empirical evaluation of our method, and Section 8 discusses about its pros and cons. Finally, Section 9 presents the related work and Section 10 summarizes with the relevant conclusions and further work.